The Number of Legal Go Positions

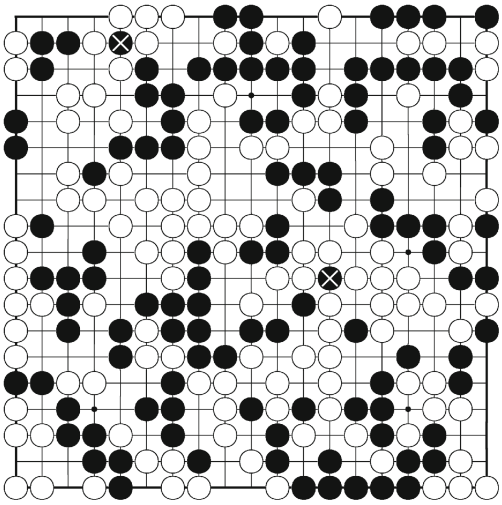
John Tromp(✉)

Stony Brook, USA
john.tromp@gmail.com

Abstract. The number of legal 19×19 Go positions has been determined as

208168199381979984699478633344862770286522453884530548425
639456820927419612738015378525648451698519643907259916015
628128546089888314427129715319317557736620397247064840935

A roughly 1.2 % fraction of the $3^{19\times19}$ total number of positions, this is more naturally expressed in ternary. Replacing the usual ternary digits 0,1,2 by +(empty), ● (black), and ○ (white) respectively, yields the following (illegal) position that counts all legal positions:



1 Introduction

Go [2,4] almost needs no introduction, but one can be found in the parent paper "Combinatorics of Go" [1], which derived a dynamic programming algorithm to compute numbers of legal positions. With the resources available at the Center for Mathematics and Computer Science (CWI) in 2006, John Tromp and Michal Koucký, who helped develop a file-based implementation, were able to count the number of legal 17×17 positions. This was announceed on August 18, 2006, over 10 months after the quick succession of results for 14×14 through 16×16.

A. Plaat et al. (Eds.): CG 2016, LNCS 10068, pp. 183–190, 2016.
DOI: 10.1007/978-3-319-50935-8_17

Since then we have been on the lookout for potential new sources of computing power for the final two steps of 18×18 and 19×19. We submitted many a proposal, both formal and informal, academic and commercial. It was not until early 2014 that Tromp got an offer from Piet Hut at the Institute for Advanced Studies, to use their computing cluster, which led to the results reported here. This paper focuses on these recent results and the software used to obtain them, at the expense of repeating a great deal of the underlying theory detailed in the parent paper. The reader is therefore strongly advised to have a copy of that paper handy for filling in some of the missing details.

2 Preliminaries

A position on an $m \times n$ Go board is a mapping from the set of *points* $\{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$ to the set of colors {empty, black, white}. Points are *adjacent* in the usual grid sense—equal in one coordinate and differing by one in the other. A point colored black or white is called a *stone*. Adjacent stones of the same color form connected components called *strings*. An empty point adjacent to a string is called a *liberty* of that string. A position can arise in a game of Go if and only if all its strings have liberties. Such positions are called *legal*. The number of legal $m \times n$ positions is denoted $L(m, n)$.

3 The Border State Graph

The parent paper established a correspondence between legal positions and paths through a graph of so-called *border states*, as illustrated in Fig. 1 for a small 3×3 board.

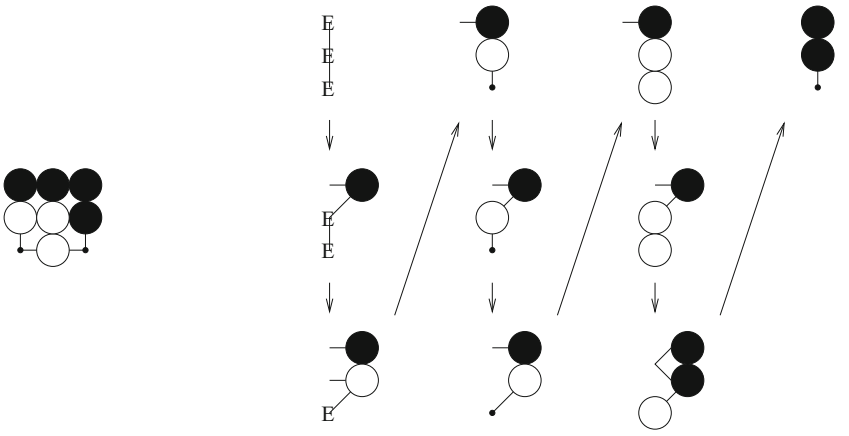


Fig. 1. A 3×3 position and corresponding path through the border state graph

We number the points of the board (plus an extra point to its right) from 0 through 9, isomorphic to the ordering of border states on the right. All the points less than a point p constitute a *partial board up to* p, and the position on these points is a called *partial position*. Each border state on the path records not only the colors of the previous $n = 3$ points ('E' denoting the board edge) but also what is needed to ensure that they have liberties when extending the partial position. This includes knowledge of which stones currently lack liberties and how they are connected in the partial position, shown as lines pointing left and possibly joining up.

The border state of a partial board up to p together with the color of p uniquely determines the *successor* border state up to $p + 1$, if legal. The *border state graph* consists of all border states and their successor transitions. An example of an illegal successor would be a white stone at $p = 3$, preventing the top left black stone from gaining a liberty. Now the problem of computing $L(m, n)$ is reduced to that of counting paths of length mn in a certain graph.

4 The Path Counting Implementation

The go counting software is publicly available at my github repository [5].

To jump right in, file `modulus.h` implicitly defines a list of relatively prime numbers each of the form $M_i = 2^{64} - d$, for many different small values of $0 \leq d < 256$. This allows us to split up the task of computing $L(m, n)$ into many smaller independent jobs that each compute modulo some M_i. The resulting set of equations

$$L(m, n) = a_i \bmod M_i,$$

is readily solved using the Chinese Remainder Theorem [3], as implemented in the Haskell program `CRT.hs`. For $L(18, 18)$, a 508 bit number, we need $\lceil \frac{508}{64} \rceil = 8$ jobs, while for $L(19, 19)$, $\lceil \frac{566}{64} \rceil = 9$ jobs suffice.

File `golegal` is a shell script for computing modular path counts, to be invoked as

```
./golegal width modulus [y [ x [incpus [memsize [height [ncpus]]]]]]
```

For example, if we want to compute $L(13, 13)$ modulo $M_1 = 2^{64} - 3$, using 3 GB of memory and 2 cores, (and we already ran `make all` to create the `start` and `legal` executables), we would run

```
./golegal 13 1 0 0 2
```

This creates a top-level directory `13.1` with data sets in subdirectories `yx.00.00` through `yx.13.00`, each one computed from the previous with multiple invocations of `legal`, one for each cpu. If problems arise necessitating a restart, then we can invoke `golegal` with appropriate values of `y` and `x`. For historical reasons, this implementation works row by row rather than column by column as in Fig. 1.

Within each `yx.*.*` subdirectory are the `start` and `end` timestamps, the `cpu.*` logs containing the standard output of all `legal` invocations, and finally

the `fromto.*.*` directories holding the actual counts. Let us look at what happens in the sample execution

```
time ./legal 13 1 12 10 2 2 1 500M &>  13.1/yx.12.11/cpu.1
```

The shell script chose a default memory footprint of 500MB, which is allocated to hold blocks of state-count pairs. The executable starts with opening all files in `13.1/yx.12.10/fromto.*.1/` directories in order to merge their already sorted records into a single stream of state-count pairs (see `instream.c`). This stream is processed in the `legal.c` code fragment

```
for (; (mb = minstream(gin))->state != FINALSTATE; nin++,deletemin(gin)) {
  sn.cnt = mb->cnt;
  nnew = expandstate(mb->state, x, newstates);
  for (i=0; i<nnew; i++) {
    sn.state = newstates[i];
    jtinsert(jts, &sn);
  }
  if (nnew < 3) // nnew == 2
    modadd(modulus, &nnewillcnt, mb->cnt);
  if (jtfull(jts))
    dumpstates(go, jts, noutfiles++, mb->state);
}
```

The call to expandstate (in `states.c`) generates the 2 or 3 successor states, each of which is paired with the state count and inserted into the custom `jtset` data structure from `sortstates.c`. State expansions involve first unpacking the highly compressed representation (using only 3 bits per border point), then trying all 3 possible colors for the next point to record the effects on liberties and connections, and packing the results back into the highly compressed representation. The sum count of missing, i.e., newly illegal, successors is maintained in `nnewillcnt` to be logged and cross-checked. Whenever the `jtset` reaches its capacity, routine `dumpstates` from `outstream.c` is called to dump the state-count pairs to files. This involves first (radix) sorting all pairs by state, merging identical ones by summing their counts, and then partitioning them over all cpus, writing one file for each.

The partition boundaries have been precomputed in `partition.c`[1] to ensure an almost uniform distribution of states over cpus. The first line of output

```
width=13 bump=11 tot=48744371 part=24372185
```

shows the width, the bump (x-coordinate) of new states, the number of states, and the boundary between states for cpu 0 and states for cpu 1. Each state pair is written as a state delta followed by the 64 bit count. With the states being sorted, the delta is just over 1 byte on average. Each file ends with a checksum record, that uses a sentinel FINALSTATE and a count such that the sum of all counts equals zero (for the given modulus).

[1] This is probably the trickiest part of the code, and was still found to contain bugs during the 18×18 run (affecting efficiency rather than correctness).

In a typical file name of `13.1/yx.12.11/fromto.1.0/1.6546124577333`, the basename consists of the number of `dumpstates` calls, followed by the next state in the input stream (in octal). This helps with mid-step restarts using manual invocations of `legal`, an advanced feature best avoided.

If the memory allocated is too small then dumpstates will be called hundreds of time, which might require thousands of files to be opened for reading in the next step, creating IO bottlenecks. For the 19×19 jobs I liked to use a minimum of 20 GB.

The final lines of output are

```
(12,10) size 24313729 xsize 24391897 mod 18446744073709551613
newillegal 8421059390853372058 needy 15106516706600782168
     legal 17975594761389357431 at (12,11)
```

The first summarizes the input stream, giving the merged size and total size in number of states, as well as reminding us of the modulus used. The next shows the sum count of illegal successors, of states with some border stones in need of liberties, and of states with no such stones.

Apart from setting up the directory structure and iterating over all the steps and cpus, the `golegal` shell script also conserves space by removing files that can be considered obsolete, and takes care to protect against accidental damage by making files and directories read-only.

The perl script `gocheck` performs many checks and balances on these numbers. For instance, the total of `newillegal` + `needy` + `legal` should be congruent to 3 times the previous step's total of `needy` + `legal`. It also checks that $L(m, n) = L(n, m)$ if the latter has been previously computed, as is usually the case when $n < m$. These checks, in addition to the file checksums make it very hard for disk/memory corruption errors to go undetected. And if any of jobs manages to produce even a slightly wrong result, then Chinese Remaindering will amplify this to a huge difference in the reconstructed result, which will then no longer match the highly accurate approximation formula (see below).

5 Results

Table 1 shows the number of legal positions for 18×18 and 19×19.

The $L(18, 18)$ computation ran from summer 2014 through March 2015, taking over 50,000 CPU-hours and 4PB of disk IO, generously provided by the Intel x86 Linux Cluster of the IAS School of Natural Sciences in Princeton. It used 8 jobs with modulo indices 1,2,3,4,5,6,7,8. The smaller of two prime factors found with Dario Alejandro Alpern's ECM implementation is 7176527950749135946361.

The 18×18 result was announced on Hacker News on March 9, 2015 [6] accompanied by a request for yet more computing power to tackle 19×19.

The $L(19, 19)$ computation ran from March 9, 2015 through December 26, 2015, taking over 250,000 CPU-hours and 30PB of disk IO, generously provided

Table 1. Number of legal $n \times n$ positions.

n	#digits	$L(n,n)$
18	153	669723114288829212892740188841706543509937780640178732810318337696945624428547218105214326012774371397184848890970111836283470468812827907149926502347633
19	171	208168199381979984699478633344862770286522453884530548425639456820927419612738015378525648451698519643907259916015628128546089888314427129715319317557736620397247064840935

by the Intel x86 Linux clusters at the IAS School of Natural Sciences in Princeton, the IDA Center for Communications Research, also in Princeton, and on a HP Helion Cloud server. It used 9 jobs with modulo indices 0,1,2,3,4,5,6,11,19. Due to delays in transferring log files, the actual reconstruction of the number didn't happen until January 20, 2016.

Factorizing $L(19,19)$ results in 8 prime factors, the first 7 of which are 5, 401, 4821637, 964261621, 2824211368611548437, 219846696500237600175961 3307922757, and 659486468368075679414404343174041 97. An interesting observation about this deconstruction is that what allows us to do this in just a few hours is that the ECM factoring algorithm is exponential, not in the number of digits itself, but in the square root thereof. Similarly, our construction of $L(19,19)$ is only possible due to the path counting algorithm being exponential, not in the number of board points, but in the square root thereof.

This final result was announced on Hacker News on January 22, 2016 [7], and has been reported on (with various inaccuracies) by the popular press [9] as well as by several enthusiast sites [8,10].

6 The Base of Liberties

If we take the mn'th root of the number of all 3^{mn} positions on an $m \times n$ board, we of course get the base of 3. If we count only legal positions, then the mn'th root can be shown to converge to some number $L < 3$. Since this single number characterizes the growth rate of stones having liberties, we call it the *base of liberties*. The parent paper showed that, conditional on some conjecture about vanishing error terms,

$$L(m,n) = A\ B^{m+n} L^{mn} (1 + O(m\phi^m))$$

for some constants A, B, $\phi < 1$, and $n = \Theta(m)$. The constants A, B, and L can all be computed as limits of expressions involving legal counts of square and almost-square boards.

$$L = \lim_{n\to\infty} \frac{L(n,n)L(n+1,n+1)}{L(n,n+1)^2},$$
$$B = \lim_{n\to\infty} \frac{L(n,n+1)}{L(n,n)L^n} = \lim_{n\to\infty} \frac{L(n,n)}{L(n,n-1)L^n},$$
$$A = \lim_{n\to\infty} \frac{L(n,n)}{B^{2n}L^{n^2}}.$$

Table 2. Legal counts of almost square boards.

n	#digits	$L(n,n+1)$
17	145	20722054276190233030395875202363901217542740727187846094339981969 33282608067036314403465202963700297341152216286750576593627459392 979397487964077
18	162	21645008927907827531439545348046842446969487357646989370951775056 32614907511229224633397451785779540083245864195480719950197794545 84564790800309660950831580481393
19	180	20020319408629769567144797301355785099698625915243038261123500773 48906207401543395415870817978902800457543055297838678738457045887 23770851289942216392403148498022616435740968427261

Of course L could also be approximated according to its definition as $L(n,n)^{n^{-2}}$ but the above formula offers much better convergence. Using the almost-square legal counts in Table 2, as computed by our algorithm, our best estimates using $L(19,19)$, $L(19,18)$, and $L(18,18)$ are

$$L \approx 2.975734192043357249381,$$
$$B \approx 0.96553505933837387,$$
$$A \approx 0.8506399258457145.$$

Table 3 shows the rapid convergence of $L(n,n)L(n+1,n+1)/L(n,n+1)^2$.

Table 3. Convergence to the base of liberties L.

n	$L(n,n)L(n+1,n+1)/L(n,n+1)^2$
15	2.97573419204335724932
16	2.975734192043357249362
17	2.9757341920433572493811
18	2.97573419204335724938097

Although the formula for $L(m,n)$ is only asymptotic, convergence turns out to be quite fast. Compared to the exact results in Table 1, it achieves relative

accuracy 0.99993 at $n = 5$, 0.99999999 at $n = 9$, and 1.00000000000023 at $n = 13$. It is consistent with all the simulated results. For $n = 99$ it gives the same result of $4 \cdot 10^{4638}$. Accuracy is also excellent far away from the diagonal. For instance, at $L(7, 268)$, the relative accuracy is still 1.0000007, witnessing the wide range of application of the asymptotic formula.

Acknowledgements. We are indebted to Piet Hut and Lee Colbert for supporting both the 18×18 and 19×19 computations, and to Michael Di Domenico for supporting and helping script the 19×19 computation.

References

1. Tromp, J., Farnebäck, G.: Combinatorics of go. In: Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 84–99. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75538-8_8
2. Wikipedia: Go (game). http://en.wikipedia.org/wiki/Go_(game)
3. Wikipedia: Chinese remainder theorem. https://en.wikipedia.org/wiki/Chinese_remainder_theorem
4. Tromp, J.: The game of Go (website). http://tromp.github.io/go.html
5. Tromp, J.: github repository. https://github.com/tromp/golegal
6. Tromp, J.: Number of legal 18×18 Go positions computed. One more to go, Hacker News, March 9, 2015. https://news.ycombinator.com/item?id=9167781
7. Tromp, J.: Number of legal Go positions computed, Hacker News, January 22, 2016. https://news.ycombinator.com/item?id=10950875
8. GoBase.org (website). http://gobase.org/
9. Johnson, L.: After 2,500 years, a Chinese gaming mystery is solved, Motherboard, January 25, 2016. http://motherboard.vice.com/read/after-2500-years-a-chinese-gaming-mystery-is-solved
10. James, M.: Number of legal Go positions finally worked out, IProgrammer, February 3, 2016. www.i-programmer.info/news/112-theory/9384-number-of-legal-go-positions-finally-worked-out.html